ASSO - Bowling

Homework 4

Guilherme de Matos Ferreira de Almeida

Masters in Informatics and Computing Engineering

29/02/2024

Contents

1 Introduction

The main goal of this paper is to explore the realm of programming by implementing three different approaches to calculating bowling scores. In this report three distinct programming paradigms will be explored: procedural[3], object-oriented[2], and pipes-and-filters[1]. Each approach offers unique insights into problem-solving and software architecture, providing valuable learning opportunities. By the end of this paper, the reader should be able to decide which paradigm suits better a software solution and the most valuable trade-offs each of these three provides.

2 Procedural Approach

The procedural approach utilizes a straightforward method to calculate bowling scores. It sequentially processes the rolls and calculates the score based on the rules of the game. This approach fits well with the problem's linear nature and simplicity. It offers a clear and direct solution to the scoring algorithm. However, it may lack the modularity and scalability of other paradigms, making maintenance and extension challenging in larger projects. In Listing 1 you can observe the simplicity behind the logic to solve this problem.

```c
#define MAX_ROLLS 21  // Maximum number of rolls in a game
#define MAX_FRAMES 10 // Maximum number of frames in a game

/**
 * @brief Calculates the score of a bowling game
 *
 * @param rolls Array of rolls in the game
 * @param num_rolls  Total number of rolls in the game
 * @return int Final score of the game
 */
int calculate_score(int *rolls, int num_rolls) {
  int score = 0;
  int roll_index = 0;

  for (int frame = 0; roll_index < num_rolls && frame <
     ↪ MAX_FRAMES; frame++) {
    if (rolls[roll_index] == 10) {  // Strike
      score += 10 + rolls[roll_index + 1] + rolls[roll_index
         ↪  + 2];
      roll_index++;
    } else if (rolls[roll_index] + rolls[roll_index + 1] ==
       ↪ 10) {       // Spare
      score += 10 + rolls[roll_index + 2];
      roll_index += 2;
    } else {                        // Open frame
      score += rolls[roll_index] + rolls[roll_index + 1];
      roll_index += 2;
    }
  }

  return score;
}
```

Listing 1: C code to calculate bowling score

3 Object-Oriented Approach

The Object-Oriented approach structures the solution around classes representing key concepts in bowling (represented in Figure 1), such as frames and the game itself. The `BowlingGame` class encapsulates the game's logic, utilizing a list of `Frame` objects to represent individual frames. Inheritance is used to enable support for specialized game types, such as `FileBasedBowlingGame`, which handles parsing the various rolls from a file. This approach promotes code reusability and modularity, with each class responsible for a specific aspect of the scoring process, promoting the principle of Separation of Concerns (SoC)[4].

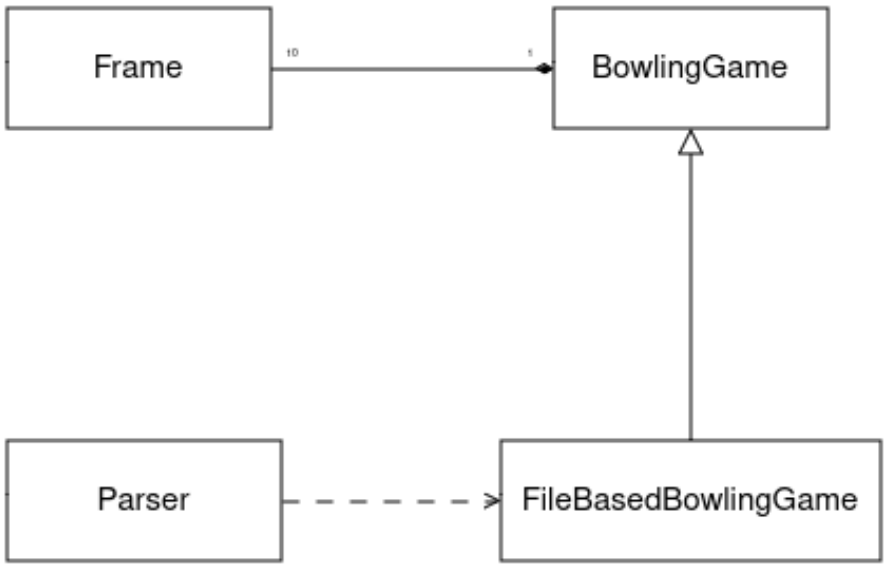

Figure 1: Class diagram of the object-oriented approach.

The `Frame` class encapsulates the rolls within each frame and provides methods to determine if a frame is a strike or a spare and to calculate a frame score.

The `BowlingGame` class orchestrates the scoring process by iterating over frames and calculating scores based on game rules. Methods such as `nextTwoRolls` and `nextRoll` handle edge cases like strikes and spares, ensuring accurate score calculation throughout the game.

The `FileBasedBowlingGame` class extends `BowlingGame` to support parsing rolls from a file, demonstrating the extensibility of this approach.

4 Pipes and Filters Approach

The pipes-and-filters approach divides the scoring process into separate filters that perform specific tasks. In this implementation, two filters are used: one for parsing rolls from input and another for calculating the score (represented in Figure 2). This approach promotes modularity and composability, allowing the reuse of individual components in various contexts. By leveraging standard input and output streams, filters can be chained together flexibly. However, this approach may introduce complexity in managing data flow and coordination between filters.

While emphasizing modularity and composability, this approach introduces some considerations regarding performance, as presented in Chapter 5.

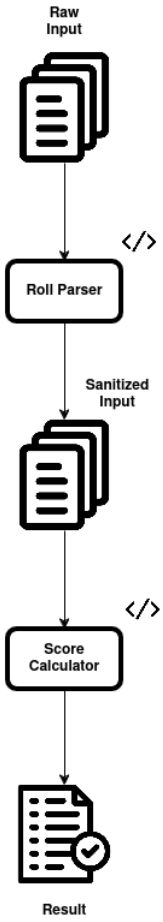

Figure 2: Pipes and filters approach's pipeline

5 Evaluation and Analysis

Each approach presents trade-offs in terms of simplicity, maintainability, and scalability. The **procedural** approach excels in simplicity but may lack modularity. **Object-oriented** design offers better organization and extensibility but introduces overhead. **Pipes-and-Filters** provide flexibility but may complicate data flow management. Choosing the most suitable approach depends on factors such as project size, requirements, and team expertise.

5.1 Performance Comparison

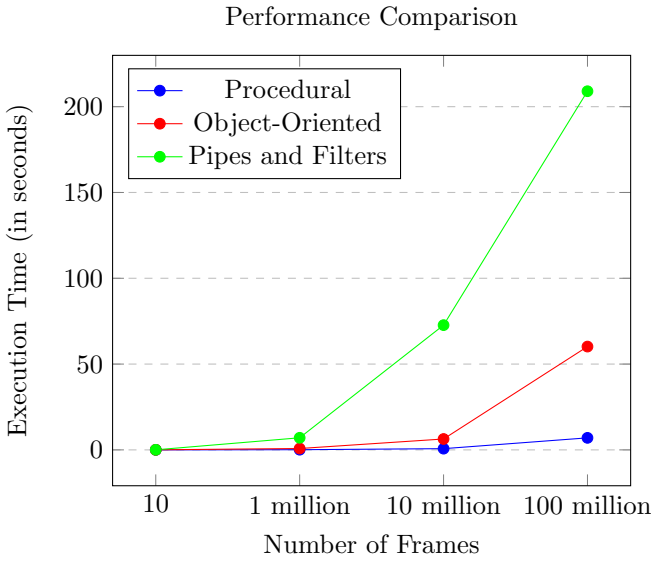

Figure 3: Performance comparison of different approaches

For this problem in particular, the approach could be chosen based on the developer's familiarity and preference, since the number of frames is fixed and doesn't scale. However, this is not the case for most real-world problems, so, I decided it would be a good exercise to change the number of frames generated (script for generating inputs in Listing 2).

The performance comparison in Figure 3 illustrates the execution time of each approach as the number of frames increases (the detailed values are present in Table 1). The **procedural** approach demonstrates low execution times consistently, while the **object-oriented** approach exhibits slightly higher times due to overhead. The **pipes-and-filters** approach shows significantly higher times, especially for larger frame counts. While in this problem, these insights are not particularly relevant, in other systems, they could help in selecting the most appropriate approach based on specific project requirements and constraints.

6 Conclusion

In conclusion, this report explores three distinct approaches to calculating bowling scores: procedural, object-oriented, and pipes-and-filters. Each approach presents unique advantages and trade-offs, providing valuable insights into software design principles.

The **procedural** approach offers simplicity in solving the problem but lacks modularity and scalability. This approach might be suitable for small-scale projects where simplicity and straightforwardness are prioritized over long-term maintainability. For example, if the bowling scoring system is a small standalone program with no plans for extensive future development, the procedural approach could be enough.

On the other hand, the **object-oriented** approach promotes code reusability and modularity through class encapsulation, although with some added complexity. This approach shines in larger projects where scalability and maintainability are crucial. For instance, if the bowling scoring system is part of a larger software ecosystem with potential future expansions, organizing the code into classes and leveraging inheritance can streamline development and facilitate future modifications.

The **pipes-and-filters** approach divides the scoring process into separate filters, offering flexibility but potentially complicating data flow management. This approach is ideal for scenarios where the scoring process needs to be highly adaptable or integrated into existing data processing pipelines. For instance, if the bowling scoring system is part of a larger data processing workflow where different components need to be easily interchangeable, the pipes-and-filters approach can provide the necessary flexibility.

Ultimately, the choice of the best approach depends on factors such as project size, requirements, and team expertise. Understanding the trade-offs between different paradigms enables informed decision-making in designing robust and scalable solutions.

References

[1] Microsoft Learn. Pipes and filters pattern. [Online; accessed 26-February-2024].

[2] Wikipedia contributors. Object-oriented programming — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/w/index.php?title=Object-oriented_programming&oldid=1209497648`, 2024. [Online; accessed 26-February-2024].

[3] Wikipedia contributors. Procedural programming — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/w/index.php?title=Procedural_programming&oldid=1202460697`, 2024. [Online; accessed 26-February-2024].

[4] Wikipedia contributors. Separation of concerns — Wikipedia, the free encyclopedia, 2024. [Online; accessed 29-February-2024].

A Appendix

A.1 Tables

Performance Comparison			
Number of Frames	**Execution Time (in seconds)**		
	Procedural	**Object-Oriented**	**Pipes-and-Filters**
10	0.000 047	0.032 679	0.038 114
1 million	0.102 647	0.796 035	7.046 469
10 million	0.710 562	6.346 300	72.694 721
100 million	7.005 621	60.199 121	209.001 709

Table 1: Performance Comparison

A.2 Code Snippets

```
NUMBER_OF_FRAMES = 100000000
FILE_NAME = f"random_{NUMBER_OF_FRAMES}_frames.txt"

def generate_score():
  first_throw = random.randint(0, 10)
  if first_throw == 10:
    return "10"

  remaining_pins = 10 - first_throw
  second_throw = random.randint(0, remaining_pins)
  return f"{first_throw} {second_throw}"

if __name__ == "__main__":
  with open(FILE_NAME, "w") as f:
    for _ in range(NUMBER_OF_FRAMES):
      score = generate_score()
      f.write(score + "\n")
```

Listing 2: Python script to generate random valid bowling scores

A.3 Project Structure

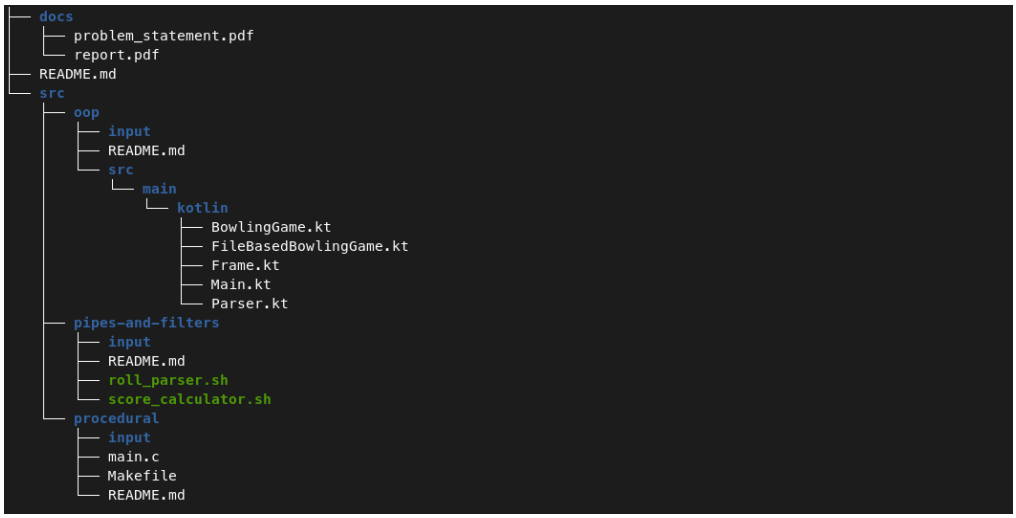

Figure 4: Project Structure

The project is organized with the following structure:

- **docs** - Contains the report and the problem statement.
- **src** - Contains the 3 developed solutions.
- **src/procedural** - Contains the procedural solution (in C).
- **src/oop** - Contains the object-oriented solution (in Kotlin).
- **src/pipes-and-filters** - Contains the pipes-and-filters solution (in Bash scripting)

Each of the solutions had a `README` file with the instructions to build, test and run the solution.

The root directory also contains a `README` file introducing the project.